📦 Add an Element at the Bottom of a Stack

Given a stack of integers and an item, insert the item at the bottom of the stack. The stack should be modified in-place without using any extra space.

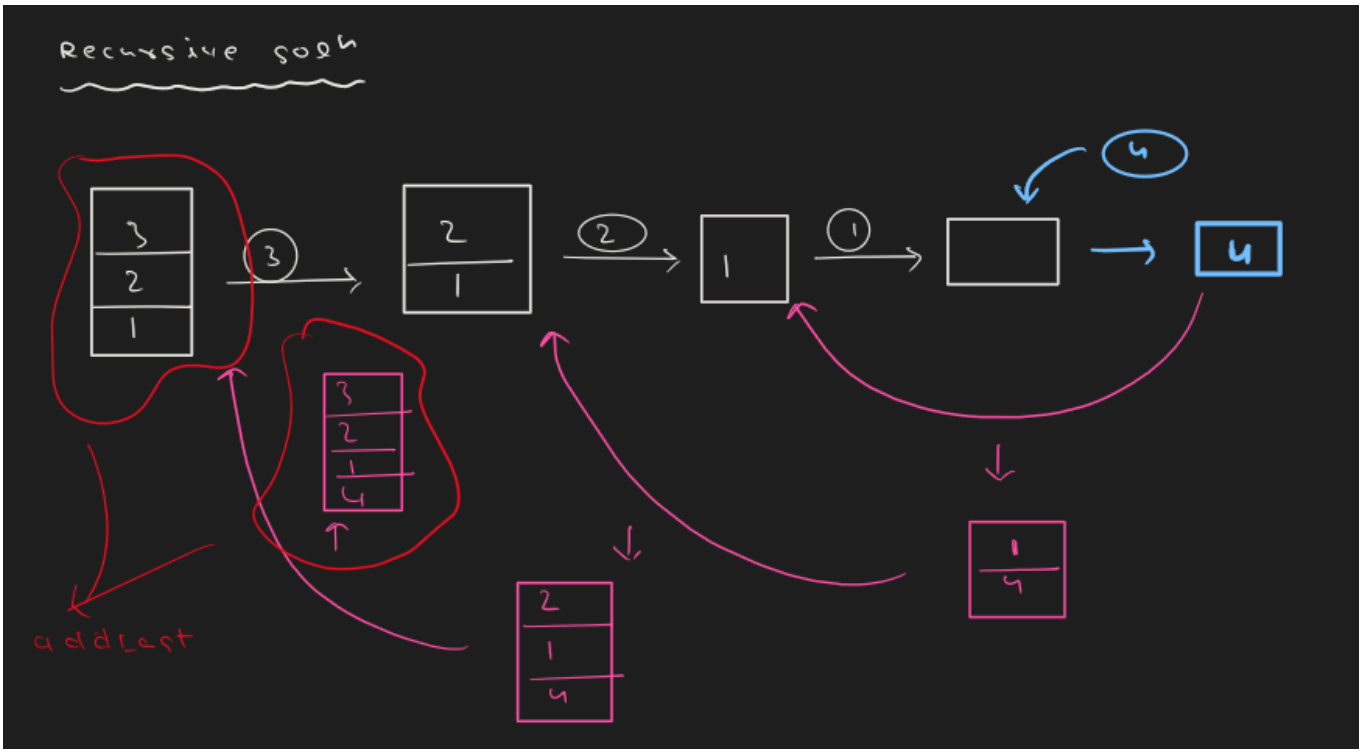
💡 Intuition

Stacks follow **Last-In-First-Out (LIFO)**, so you can't directly insert an element at the **bottom**.
To do this, you need to temporarily **remove all elements**, insert the new item, and then **put everything back** in the same order.

🛠️ Solution (Approach)

1. If the stack is empty, push the new item — this becomes the bottom.
2. Otherwise:
    - Pop the top element.
    - Recursively call the function to reach the bottom.
    - Push the popped element back (restoring order).

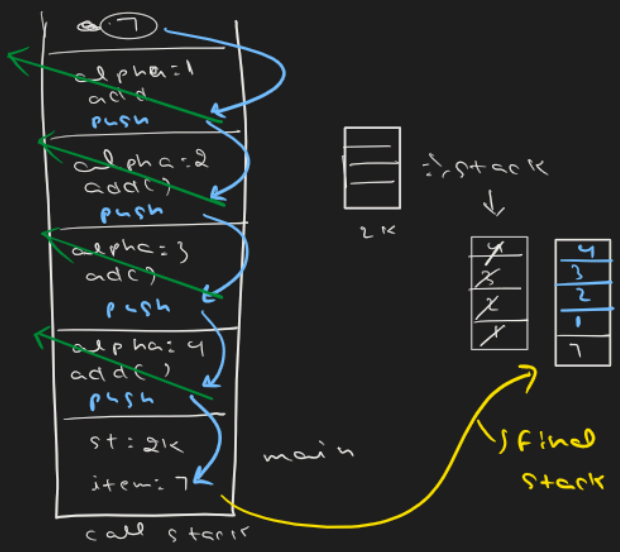
This uses the **call stack** (function recursion) to reverse and rebuild the stack around the inserted item.




```java
public class AddAtLastInStack {
	public static void main(String[] args) {
		Stack<Integer> st = new Stack<>();
		st.push(10);
		st.push(20);
		st.push(30);
		st.push(40);
		st.push(50);
		System.out.println(st);

		addLast(st, 7);
		System.out.println(st);
	}

	private static void addLast(Stack<Integer> st, int item) {
```

```
            if (st.isEmpty()) {
                  st.push(item);
                  return;
            }
            int x = st.pop();
            addLast(st, item);
            st.push(x);
      }
}
```

```
public static void addLast(Stack<Integer> st, int item){
    if(st.isEmpty()){
        st.push(item);
        return;
    }
    int alpha = st.pop();
    addLast(st, item);
    st.push(alpha);
}
```



🔁 Reverse a Stack

Given a stack of integers, reverse the entire stack in-place. The top element should become the bottom one, and the bottom should become the top — without using any extra space.

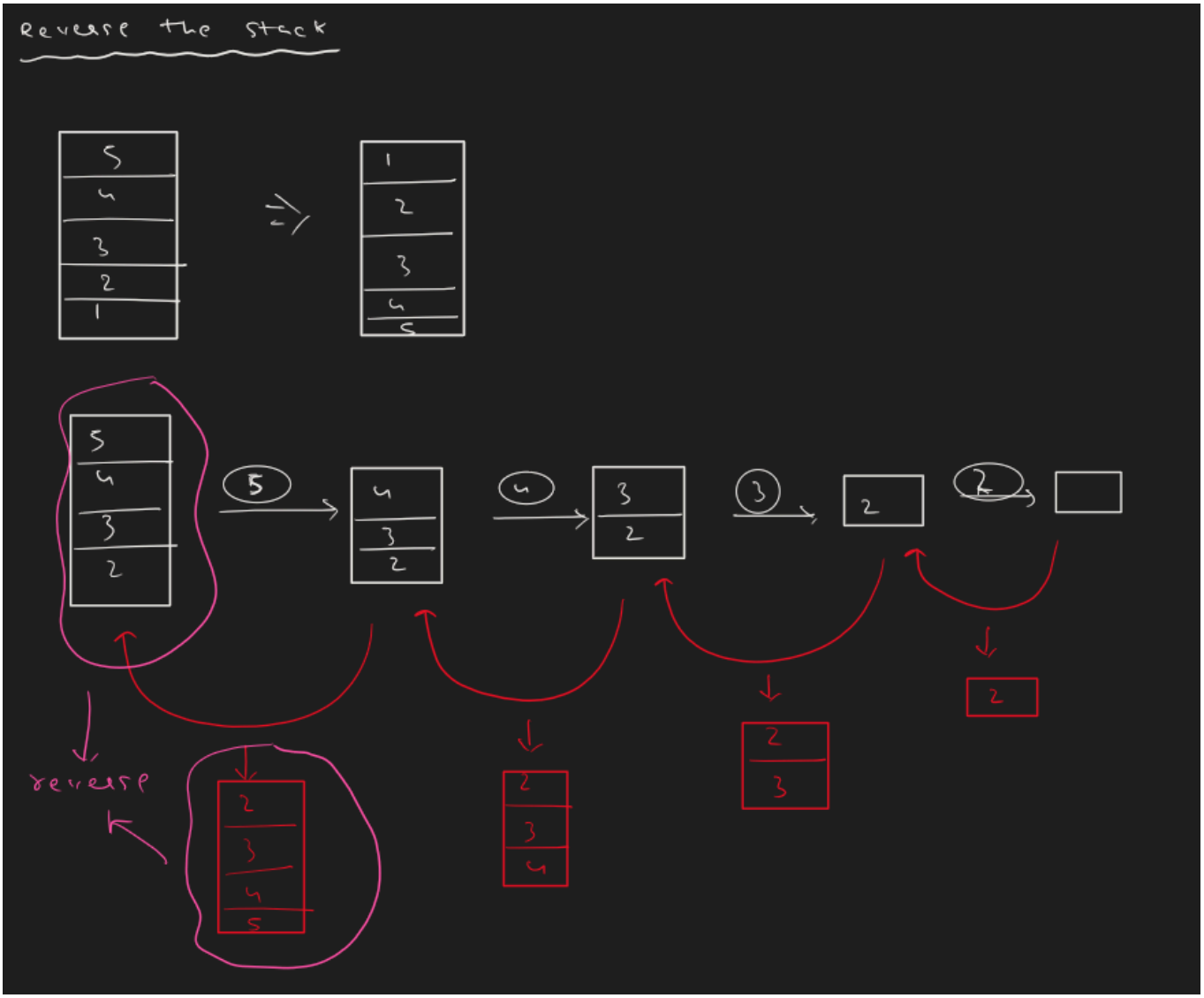
💡 Intuition

Stacks follow **Last-In-First-Out (LIFO)**, so reversing their order directly isn't possible using only standard stack operations.
To reverse the stack, we **remove all elements recursively**, and then **insert each element at the bottom** as we return from the recursion. This inverts the original order.

🛠️ Solution (Approach)

1. If the stack is empty, return (base case).
2. Pop the top element.
3. Recursively reverse the remaining stack.
4. Insert the popped element at the bottom using the `addLast()` function.

This leverages the **call stack** to temporarily hold elements, and rebuilds the original stack in reverse order.




```java
public class ReverseStack {
	public static void main(String[] args) {
		Stack<Integer> st = new Stack<>();
		st.push(10);
		st.push(20);
		st.push(30);
		st.push(40);
		st.push(50);
		System.out.println(st);

		reverse(st);
		System.out.println(st);
	}

	private static void reverse(Stack<Integer> st) {
		if (st.isEmpty()) {
			return;
		}
		int x = st.pop();
		reverse(st);
		// st.push(x);
		addLast(st, x);
	}

	private static void addLast(Stack<Integer> st, int item) {
```

```
            if (st.isEmpty()) {
                  st.push(item);
                  return;
            }
            int x = st.pop();
            addLast(st, item);
            st.push(x);
      }

}
```

🎯 Construct Smallest Number From DI String

https://leetcode.com/problems/construct-smallest-number-from-di-string/

🧠 Trick to construct the smallest number from a DI pattern:

1. **Use numbers from 1 to n+1** (if pattern has length `n`).
2. **Scan left to right**.
3. **Whenever you hit an `'I'` or end of pattern**, fill numbers **in reverse** from the last `'I'` or start.
    - That is, for each group ending at `'I'` (or end), fill the group backwards with increasing numbers.
    - This naturally places larger numbers to the left in a `'D'` chain and smaller ones at the end (which satisfies `'D'`, then `'I'`).

🧩 How to visualize:

Say you have: `"D D I D I"`

1. Count is 5 → use numbers `1` to `6`.
2. We look for increasing groups ending at `'I'` or end.
3. For each group, we assign numbers in **reverse**.

Pattern	Positions	Fill from
`D D I`	positions 0–2	fill 3..1
`D I`	positions 3–4	fill 5..4
End	position 5	fill 6

➡️ Final result: `3 2 1 5 4 6`

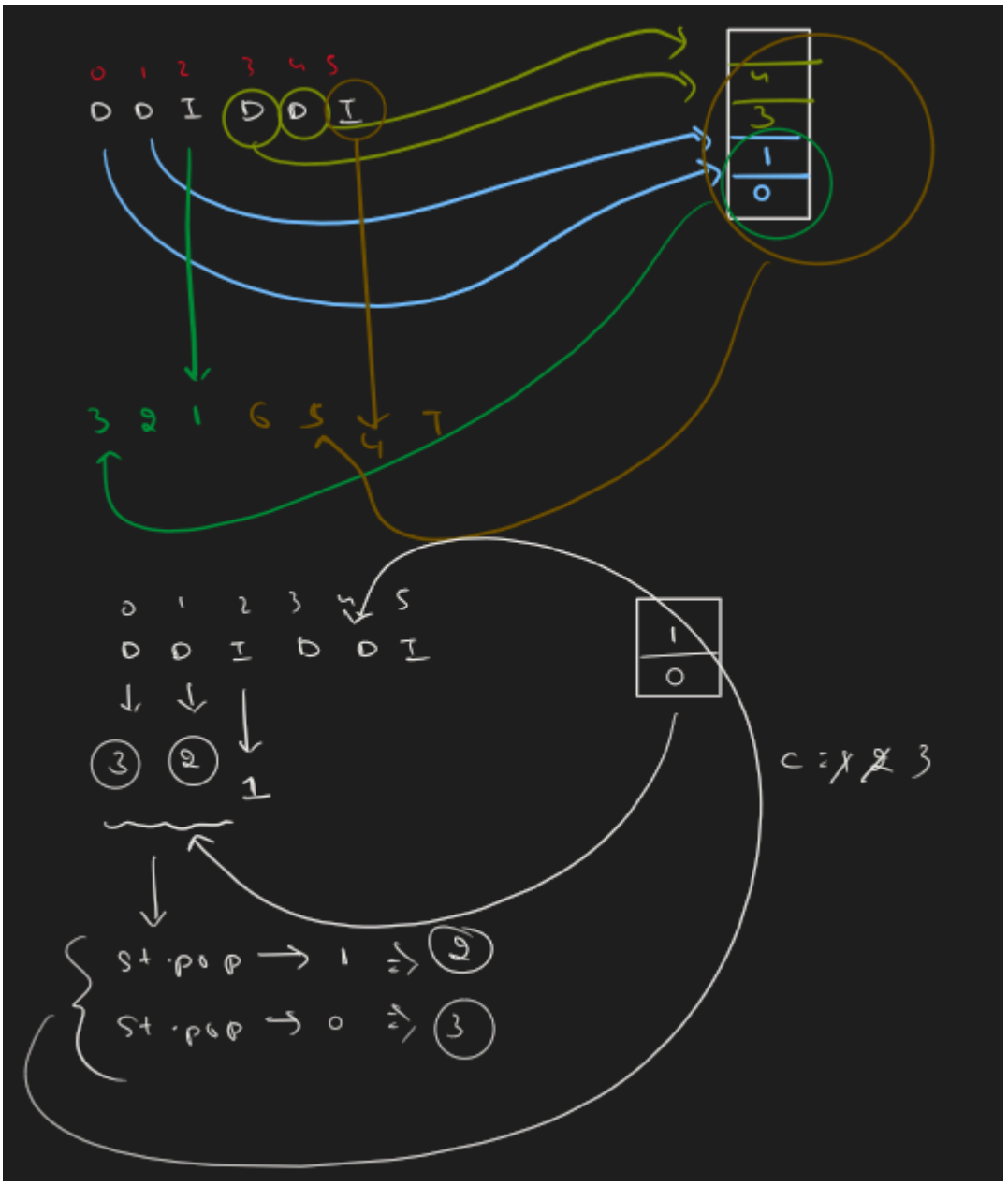
✅ Rule of thumb:

- For every sequence of `'D'`, delay filling until you hit an `'I'` or end.

- Then fill that group in reverse using the next available numbers.

🧠 **Stack Intution**:

- Push the next smallest number for each position as you go.
- When you see a `'D'`, keep pushing (delay assigning) to build a descending sequence.
- When you hit an `'I'` or reach the end, pop all numbers from the stack and append them — this reverses the delayed numbers, creating the correct descending order.
- This way, the stack helps you easily reverse descending parts while keeping the sequence smallest lex order.



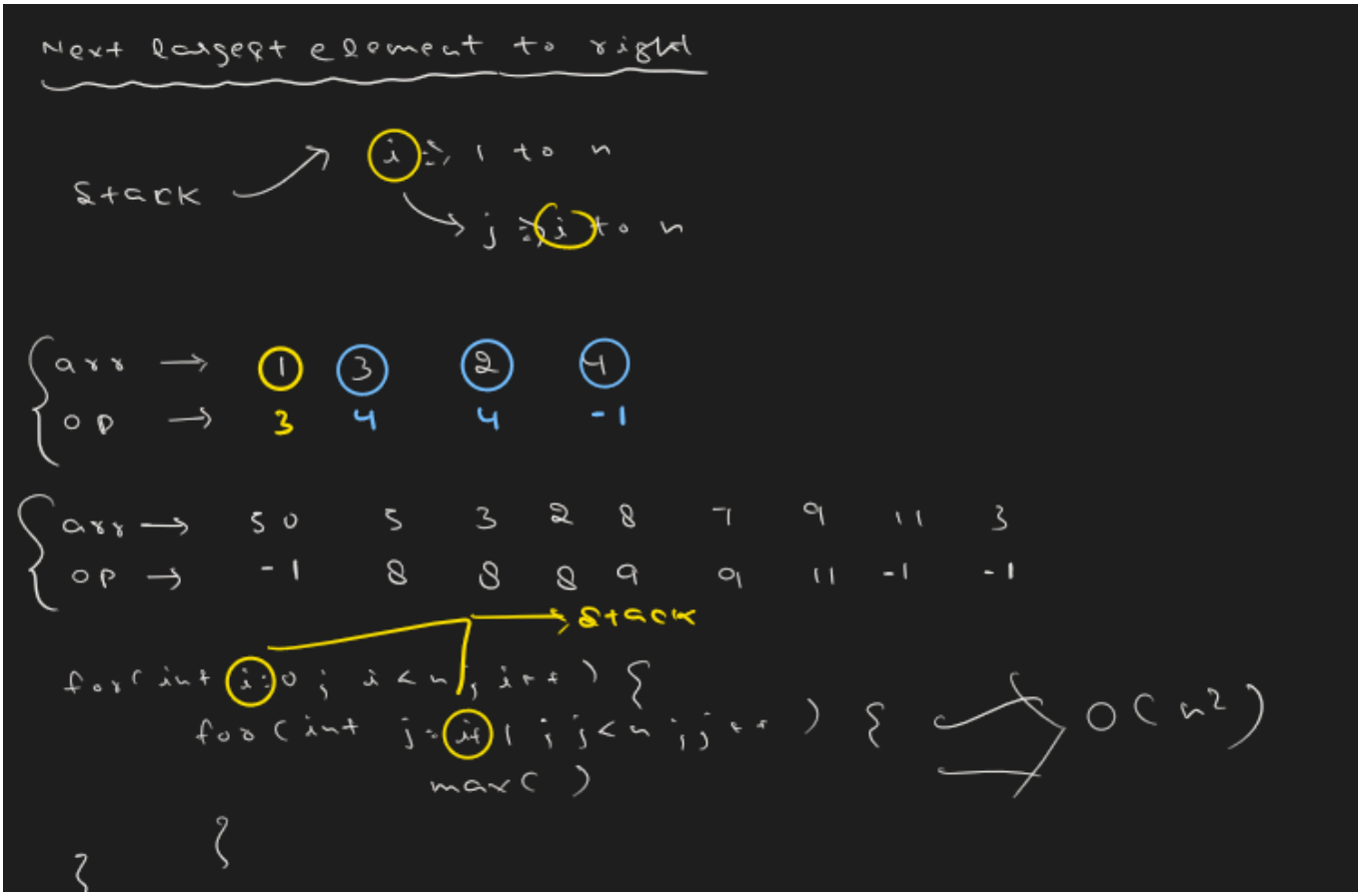
```java
public class ConstructSmallestNumberFromDIString {
	public static void main(String[] args) {
		String str = "IIIDIDDD";
		System.out.println(constructSmallestNumber(str));
	}

	public static String constructSmallestNumber(String str) {
		Stack<Integer> st = new Stack<>();
		int[] ans = new int[str.length() + 1];
		int c = 1;
```

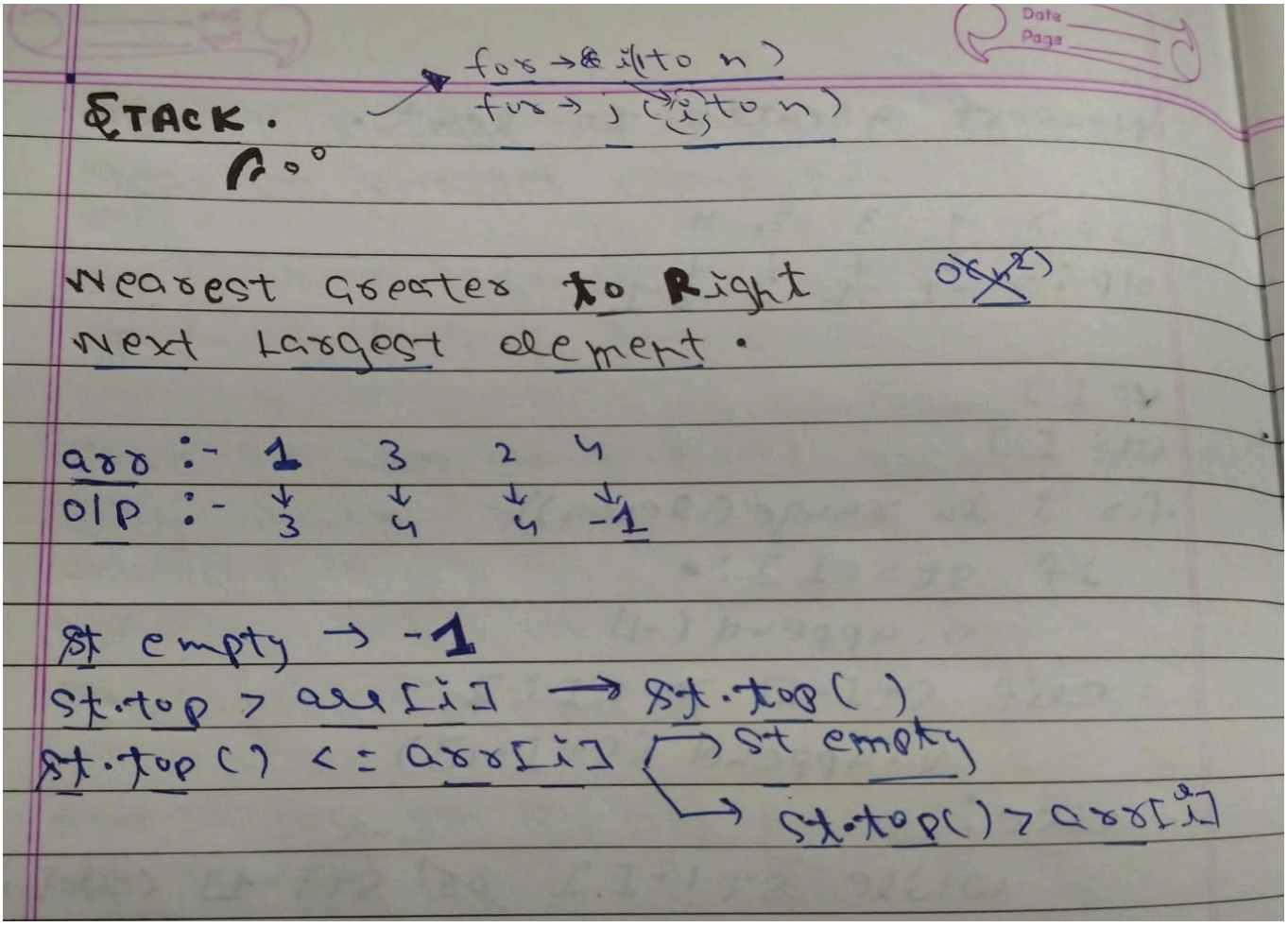
```
            for (int i = 0; i <= str.length(); i++) {
                  if (i == str.length() || str.charAt(i) == 'I') {
                        ans[i] = c;
                        c++;
                        while (!st.isEmpty()) {
                              ans[st.pop()] = c;
                              c++;
                        }
                  } else {
                        st.push(i);
                  }
            }
            String s = "";
            for (int i = 0; i < ans.length; i++) {
                  s = s + ans[i];
            }
            return s;
      }

}
```

```
class Solution {
    public String smallestNumber(String str) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[str.length() + 1];
        int c = 1;
        for (int i = 0; i <= str.length(); i++) {
            if (i == str.length() || str.charAt(i) == 'I') {
                ans[i] = c;
                c++;
                while (!st.isEmpty()) {
                    ans[st.pop()] = c;
                    c++;
                }
            } else {
                st.push(i);
            }
        }
        String s = "";
        for (int i = 0; i < ans.length; i++) {
            s = s + ans[i];
        }
        return s;
    }
}
```

🎯 Next Greater Element

https://codeskiller.codingblocks.com/problems/1053





```java
public class NextGreaterElementRight {
      public static void main(String[] args) {
            int[] arr = { 4, 5, 2, 10 };
            int[] ans = nger(arr);
            display(arr);
            display(ans);
      }

      public static int[] nger(int[] arr) {
            Stack<Integer> st = new Stack<>();
            int[] ans = new int[arr.length];
            for (int i = arr.length - 1; i>=0; i--) {
            if(st.isEmpty()){
                ans[i] = -1;
            }
            else if(st.peek() > arr[i]){
                ans[i] = st.peek();
            }
            else{
                while (!st.isEmpty() && st.peek() <= arr[i]) {
                            st.pop();
                      }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
                  st.push(arr[i]);
            }
            return ans;
      }

    public static void display(int[] a){
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.println();
    }

}
```

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
```

```
        }
        int[] ans = nger(arr);
        // display(arr);
        display(ans);
    }


    public static int[] nger(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = arr.length - 1; i>=0; i--) {
            if(st.isEmpty()){
                ans[i] = -1;
            }
            else if(st.peek() > arr[i]){
                ans[i] = st.peek();
            }
            else{
                while (!st.isEmpty() && st.peek() <= arr[i]) {
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(arr[i]);
        }
        return ans;
    }

    public static void display(int[] a){
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        // System.out.println();
    }

}
```

🎯 Next Greater Element Left

```
public class NextGreaterElementleft {
      public static void main(String[] args) {
            int[] arr = { 4, 5, 2, 10 };
            int[] ans = ngel(arr);
            display(arr);
            display(ans);
      }

      public static int[] ngel(int[] arr) {
```

```
            Stack<Integer> st = new Stack<>();
            int[] ans = new int[arr.length];
            for (int i = 0; i<arr.length; i++) {
            if(st.isEmpty()){
                ans[i] = -1;
            }
            else if(st.peek() > arr[i]){
                ans[i] = st.peek();
            }
            else{
                while (!st.isEmpty() && st.peek() <= arr[i]) {
                            st.pop();
                      }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
                  st.push(arr[i]);
            }
            return ans;
      }

    public static void display(int[] a){
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
            }
        System.out.println();
    }

}
```

🎯 Next Greater/Smaller Element Left/Right

```
public class Main {
      public static void main(String[] args) {
      int[] arr = {4, 5, 2, 10, 12, 7};
      int[] ans1 = nger(arr);

      display(arr);
      display(ans1);

      int[] ans2 = ngel(arr);
      display(ans2);

      int[] ans3 = nler(arr);
      display(ans3);

      int[] ans4 = nlel(arr);
      display(ans4);
      }
```

```
public static int[] nger(int[] arr){
Stack<Integer> st = new Stack<>();
int[] ans = new int[arr.length];

for(int i=arr.length-1; i>=0; i--){
      if(st.isEmpty()){
      ans[i] = -1;
      }
      else if(st.peek() > arr[i]){
            ans[i] = st.peek();
      }
      else{
            while(!st.isEmpty() && st.peek() <= arr[i]){
            st.pop();
            }
            if(st.isEmpty()){
            ans[i] = -1;
            }
            else{
            ans[i] = st.peek();
            }
      }
      st.push(arr[i]);
}
return ans;
}

public static int[] ngel(int[] arr){
Stack<Integer> st = new Stack<>();
int[] ans = new int[arr.length];

for(int i=0; i<arr.length; i++){
      if(st.isEmpty()){
      ans[i] = -1;
      }
      else if(st.peek() > arr[i]){
            ans[i] = st.peek();
      }
      else{
            while(!st.isEmpty() && st.peek() <= arr[i]){
            st.pop();
            }
            if(st.isEmpty()){
            ans[i] = -1;
            }
            else{
            ans[i] = st.peek();
            }
      }
      st.push(arr[i]);
}
return ans;
}
```

```
public static int[] nler(int[] arr){
Stack<Integer> st = new Stack<>();
int[] ans = new int[arr.length];

for(int i=arr.length-1; i>=0; i--){
      if(st.isEmpty()){
      ans[i] = -1;
      }
      else if(st.peek() < arr[i]){
            ans[i] = st.peek();
      }
      else{
            while(!st.isEmpty() && st.peek() >= arr[i]){
            st.pop();
            }
            if(st.isEmpty()){
            ans[i] = -1;
            }
            else{
            ans[i] = st.peek();
            }
      }
      st.push(arr[i]);
}
return ans;
}

public static int[] nlel(int[] arr){
Stack<Integer> st = new Stack<>();
int[] ans = new int[arr.length];

for(int i=0; i<arr.length; i++){
      if(st.isEmpty()){
      ans[i] = -1;
      }
      else if(st.peek() < arr[i]){
            ans[i] = st.peek();
      }
      else{
            while(!st.isEmpty() && st.peek() >= arr[i]){
            st.pop();
            }
            if(st.isEmpty()){
            ans[i] = -1;
            }
            else{
            ans[i] = st.peek();
            }
      }
      st.push(arr[i]);
}
return ans;
}
```

```
	public static void display(int[] a){
	for(int i=0; i<a.length; i++){
		System.out.print(a[i] + " ");
	}
	System.out.println();
	}

}
```